IMPLEMENTATION OF A RATFOR PREPROCESSOR

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
RADHA RAMAN GARGEYA

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST, 1978

CERTIFICATE

This is to certify that the thesis entitled, "Implementation of a Ratfor Preprocessor", is a record of the work carried out under our supervision and that it has not been submitted elsewhere for a degree.

M.S. KRISHNAMOORTHY
Assistant Professor

S.N. MAHESHWARI
Assistant Professor

Computer Science Programme
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

Kanpur
August 1978.

TABLE OF CONTENTS

ABSTRACT

Ratfor (short form of RATional FORtran) is a programming language designed by Kernighan and Plauger. Ratfor extends the control structures of Fortran to facilitate structured programming. Ratfor also supports some features aimed at making a program concise and readable. A description of the language is given. A preprocessor, which accepts the Ratfor source program as the input and produces an equivalent Fortran program, is designed and implemented. A scheme for integrating the preprocessor with the IBM 7044/1401 system is given. Lastly, areas for further work are identified.

CHAPTER 1

INTRODUCTION

The recent developments in hardware technology, and the advances in machine architecture have certainly highlighted the fact that the software costs from the dominant component of computing costs. It has been recognized that one way of reducing the software costs is by adopting a language designed to help in improving the existing programming methodologies and programming supporting tools.

The concept of structured programming is of relevance here. A consensus has been reached[9] about the control structures a programming language should have to support structured programming. Efforts have been made to provide additional control structures to the already existing programming languages. Such extensions for Fortran have been carried out and there are around fifty structured versions of Fortran available [12]. Kernighan and Plauger [8] have suggested yet another simple extension of Fortran, known as Ratfor (short form of RATional FORtran).

The aim of this thesis is to implement Ratfor. A convenient way of doing this is through a preprocessor. The preprocessor accepts the Ratfor source program as its input, and translates it into an equivalent Fortran program.

This Fortran code can be compiled in the normal manner by any available Fortran compiler.

Chapter 2 describes the Ratfor language with sample programs. Few minor modifications made in the language due to implementational constraints are also given.

Chapter 3 discusses in detail the design strategy adopted in developing the preprocessor. Details regarding the data structures used, the algorithm employed, the code generation rules, along with brief explanations of few important routines are included.

Chapter 4 identifies those areas where further related work can be pursued.

CHAPTER 2

RATFOR DIGEST

The primary purpose of Ratfor, as laid down in Chapter 1, is to make Fortran a better programming language, for writing well-structured programmes. This is done by providing the control structures that are unavailable in bare Fortran.

2.1 A Synopsis:

A context free grammar for Ratfor is given below in the Backus-Naur Form (BNF) [8].

```
<program>     ::= <statement>   <program> <statement>
<statement>   ::=  if( <condition> ) <statement>
                 | if( <condition> ) <statement> else<statement>
                 | while( <condition> ) <statement>
                 | for( <initialize> ; <condition> ;
                        <reinitialize> ) <statement>
                 | repeat <statement>
                 | repeat <statement>  until( <condition> )
                 | do <limits>  <statement>
                 |<digits> <statement>
                 | next
                 | break
                 | <program>
                 | other
```

The above BNF specification is simple and straightforward. The keywords of Ratfor are if, else, for, repeat, until, do, next, and break, which represent the control flow constructs. <statement> is any legal statement in Fortran: assignment, declaration, subroutine case, I/O etc., or any of the Ratfor statements described. Any Fortran or Ratfor statement or group of these can be enclosed in braces { } to make it into a compound statement, which is then equivalent to a single statement.

<condition> is identical to a valid logical expression of Fortran and <limits> to the DO loop control expression of Fortran. <initialize> and <reinitialize> in the for statement can be any assignment statements involving a variable which is initialized and later updated during each iteration of the for statement. <digits> stands for a string of digits, that is, a valid Fortran statement number.

Type other is an important specification, for it frees Ratfor from having to know very much about Fortran. A statement which does not begin with one of the keywords (or <digits> or a {), must be an other.

Ratfor has the character declaration. In most environments this will be synonymous with integer. It is good to distinguish the two types of variables in all the programs, so that one can tell immediately what the usage of a particular variable will be.

Ratfor has been designed to make programs concise and readable. It is free-form: a Ratfor source statement, most of which is written in the lower case alphabet, may appear anywhere on a line. It is important to indent systematically so as to discern the nesting of control flow in the program.

Ratfor uses == for the equality test .EQ. and ¬= for the inequality .NE. . The symbols ¬ , | , and & stand for the logical operators, .NOT., .OR., and .AND. respectively. The relational operators < , <=, > , and >= have the obvious meanings of .LT., .LE., .GT., and .GE. respectively.

Generally the end of a line marks the end of a statement. But constructions like

```
if(c¬== NEWLINE)
      linect = linect + 1
```

are obviously not finished after the line that contains the <u>if</u> and so they are continued automatically. This is also true of conditions which extend over more than one line, as in

```
while (c == BLANK
       |c == TAB
       |c == NEWLINE)
          i = i + 1
```

Lines ending with a comma are also continued. Multiple statements, separated by a semicolon, may appear on a single line.

A sharp sign # anywhere in a line signals the beginning of a comment, which is terminated by the end of the line. This comment convention is more flexible than Fortran's "C in column one" because comments and code can co-exist on the same line. Ratfor also allows symbolic constants which contribute a great deal to the readability of the code. Upper case characters are used to make the symbolic constants conspicuous. A Ratfor identifier, i.e., a variable name, can be of any length. A decision on the actual identifier length is left to the implementation stage.

An arbitrary Fortran program is not necessarily a Ratfor program. Blanks are significant in Ratfor in that keywords like, _if_, symbolic constants like NEWLINE and relationals like >= must not contain blanks or they will not be recognized. Furthermore, keywords are reserved, and should not be used as variable names. Standard Fortran comments, continuation conventions and the arithmetic IF are incompatible, but since Ratfor provides better alternatives for each, this is not a serious problem.

2.2 <u>Control Flow Constructs</u>:

The control flow constructs of Ratfor are briefly described here. These are the features that impart to

Ratfor its specific significance. Few of these are explained through flowcharts in Fig.2.1.

2.2.1 IF Statement:

Syntax: if(<condition>)
<statement1>
else <statement2>

Semantics: This means that if the <condition> is true, execute <statement1> ; and if the <condition> is false, execute <statement2> . The else part is optional.

Notes: As in most languages, the construction

```
if( <condition1> )
      if( <condition2> )
            <statement1>
      else
            <statement2>
```

is ambiguous - the else can be associated with either if. Braces can be used to clarify this. In the absence of braces, each else is associated with the closest proeding if. The example above is indented to agree with the binding rule, but it is advisable to use braces in such cases, to make the intent perfectly clear.

2.2.2 WHILE Statement:

Syntax: while(<condition>)
<statement>

Semantics: The <condition> is tested. If the outcome of the test is true, then the <statement> is executed, and the while statement is reentered. If the tested <condition> is false, the execution of the while statement is complete.

2.2.3 FOR Statement:

Syntax: for(<initialize>;<condition>;<reinitialize>)
<statement>

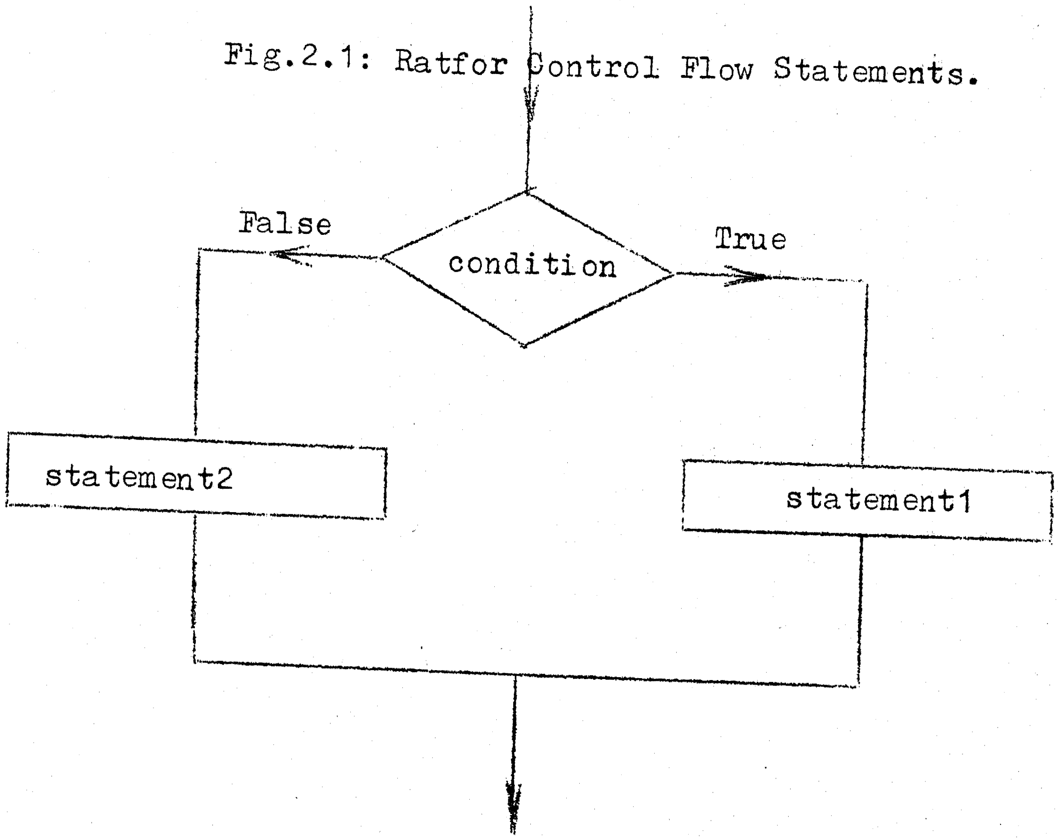
Semantics: This is equivalent to

```
<initialize>
while( <condition> )
        <statement>
        <reinitialize>
```

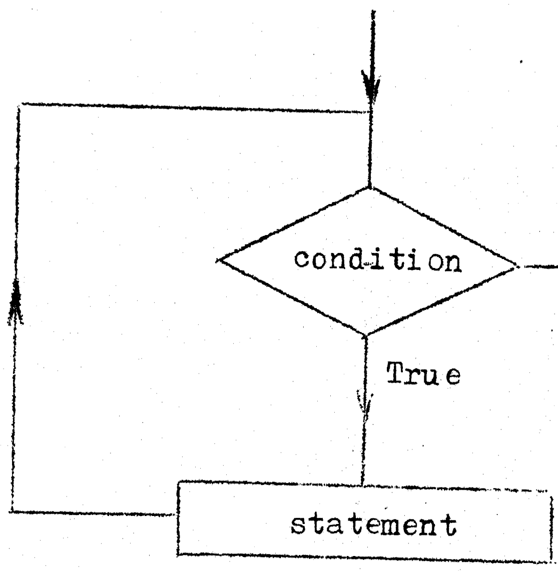
with one exception given below. The <initialize> and <reinitialize> parts are single Fortran statements. If either of them is omitted, the corresponding part of the expansion is also omitted. If the <condition> is omitted, it is taken to be always true, resulting in an 'infinite' loop.

Notes: The purpose of <initialize> and <reinitialize> is to provide loop control.

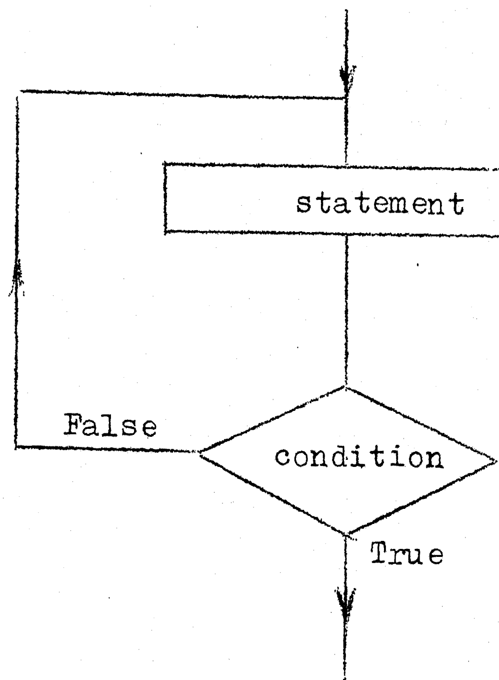
Fig.2.1: Ratfor Control Flow Statements.



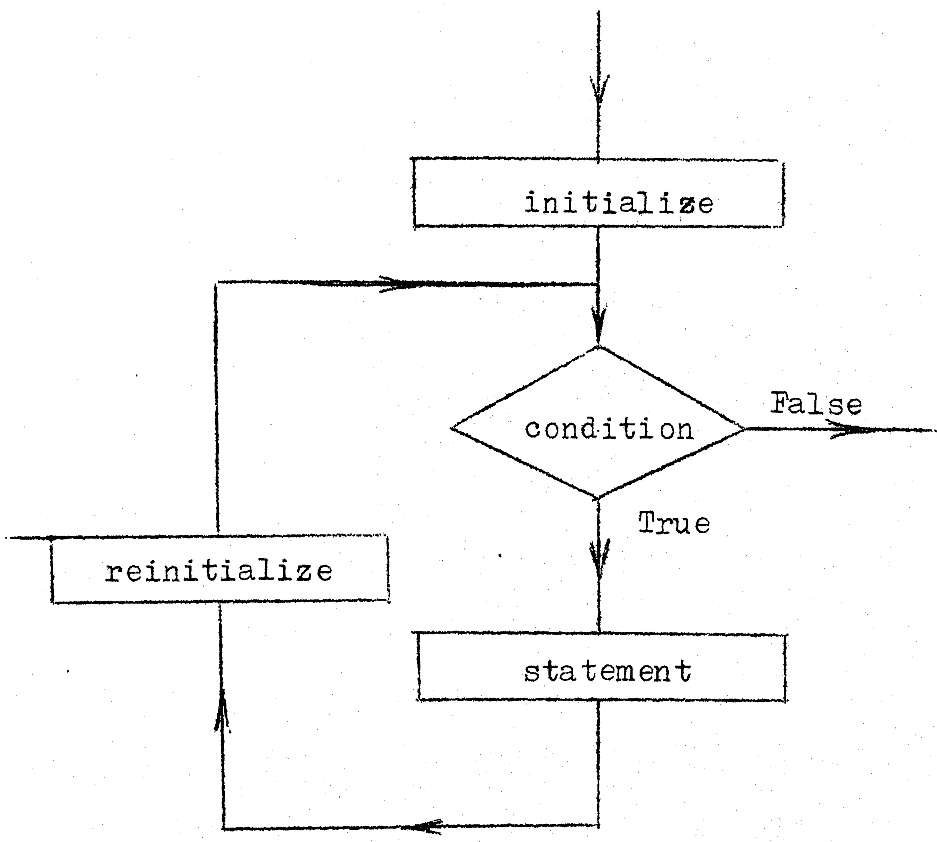
(a) Simple <u>if</u>-<u>else</u> Statement.



(b) <u>while</u> statement.



(c) repeat-until Statement



(d) for Statement.

2.2.4 REPEAT-UNTIL Statement:

Syntax: <u>repeat</u>

<statement>

<u>until</u>(<condition>)

Semantics: The <statement> is executed one or more times until <condition> becomes true, at which time the loop is exited. The <u>until</u> part is optional; if it is omitted, the result is an infinite loop, which must be broken some other way.

2.2.5 DO Statement:

Syntax: <u>do</u> <limits> <statement>

Semantics: The <u>do</u> statement sets up a standard Fortran DO loop. <limits> must be a legal loop control statement of Fortran, like I = 1,N.

2.2.6 BREAK Statement:

Syntax: <u>break</u>

Semantics: The <u>break</u> statement is one way to get out of an infinite loop. It causes whatever loop it is contained in (i.e. <u>while</u>, <u>for</u>, <u>repeat</u>, or <u>do</u>) to be exited immediately. Only the loop that immediately encloses a <u>break</u> is terminated.

2.2.7 NEXT Statement:

Syntax: <u>next</u>

Semantics: The _next_ statement causes whatever loop it is contained in to go immediately to the next iteration, skipping the rest of the loop body. _next_ goes to the condition of a _while_, _do_, or _until_, to the first statement of an infinite _repeat_ loop, and to the <reinitialize>of a _for_.

2.3 Macroprocessing

Occurrence of the macro names (with actual parameters, wherever necessary) in the text of Ratfor program causes textual substitution of this name by the body of the macro. Macro definitions may be parenthesised. As an example of simple macro definition, consider

define(EOF, -1)

The name of the macro is EOF. The body of the macro is the string "-1". Macros can also have arguments, e.g.,

define(putc(c),putch(c,STDOUT))

The name of the macro is putc(x). The body of the macro is the string "putch(x,STDOUT)". And x is the formal parameter.

2.4 Minor Modifications

The limitations of the available character set in IBM 7044/1401 necessitated minor modifications in the original description of the language.

The logical operators denoted by ¬, | , and & are represented by the Fortran equivalents, .NOT., .OR., and .AND. respectively. Similar is the case with the relational operators,

like =, =, etc. which are also expressed in the usual Fortran manner by .LE., and .GE., etc. respectively. It is possible to use only one relational operator == (equals to) without any substitution. But to maintain uniformity, that is also denoted by .EQ., semicolon, ";", is replaced by ".,", and the delimiter for the comment, "# ", by"/*".

Since the lower case alphabet also is not available, the Ratfor source statements are written in the upper case alphabet only . As and when a suitably **extended** character set is made available, all these features which enhance the readability of the Ratfor code can be implemented with little effort.

2.5 <u>Sample Programs</u>:

The features of the Ratfor language are illustrated by the two sample programs given below. The first program sorts an integer array of size n, by employing the shell sort algorithm. The second program converts an integer to a character string.

2.5.1 Shell Sort Program:

The basic idea of the shell sort is that in the early stages far-apart elements are compared, instead of adjacent ones. This tends to eliminate large amounts of disorder quickly. Gradually the interval between the compared elements is decreased, until it reaches one, at which

point it effectively becomes an adjacent interchange method.

```
# Shell - Shell sort v(1) ... v(n) increasing
    subroutine shell (v,n)
    integer gap, i, j, jg, k, n, v(n)

    for(gap = n/2; gap > 0; gap = gap/2)
        for(i = gap+1; i <= n; i = i+1)
            for(j = i-gap; j > 0; j = j-gap) {
                jg = j+gap
                if(v(j) <= v(jg)) # compare
                    break
                k = v(j)          #
                v(j) = v(jg)      # exchange
                v(jg) = k         #
                }
        return
        end
```

The outermost loop controls the gap between compared elements. Initially n/2, it shrinks by a factor of two each pass until it becomes zero. The middle loop compares elements separated by 'gap'; the innermost loop reverses any that are out of order. Since 'gap' is eventually reduced to one, all elements are ordered correctly.

2.5.2 Integer to Character String Conversion:

This function subprogram converts an integer to characters in an array provided by the calling program, and returns the number of characters it took. The digits are obtained in reverse order, and flipped before returning.

```
# itoc - convert integer 'int' to character string in 'str'
    integer function itoc(int, str, size)
    integer abs, mod
    integer d, i, int, intval, j, k, size
    character str(size)
    string digits "0123456789"

    intval = abs(int)
    i = 0
    repeat {                                    # generate digits
            i = i+1
            d = mod(intval,10)
            str(i) = digits(d+1)
            intval = intval/10
            }until(intval == 0 | i >= size)
    if(int < 0 & i < size) {                   # then sign
            i = i+1
            str(i) = MINUS
            }
    itoc = i-1
```

```
for(j = 1; j < i; j = j+1) {          # then reverse
        k = str(i)
        str(i) = str(j)
        str(j) = k
        i = i-1
        }
return
end
```

abs is a function that returns the absolute value of its argument. The string declaration used in the above program is implemented by means of a macro. For instance, the declaration

```
string id "iit"
```

is expanded into standard Fortran as follows:

```
INTEGER ID(3)
DATA ID(1)/LETI/
DATA ID(2)/LETI/
DATA ID(3)/LETT/
```

LETI, LETT, and MINUS are symbolic constants.

Note: A major portion of this chapter that describes the Ratfor language is of a collatory nature from the book "Software Tools" by Kernighan and Plauger [8].

CHAPTER 3

DESIGN AND IMPLEMENTATION OF THE PREPROCESSOR

Ratfor is a structured extension of Fortran. The advantages of implementing Ratfor through a preprocessor are [14]

1) the specification of Ratfor can be defined independently of Fortran,
2) programs written in Fortran can be freely linked with programs written in Ratfor,
3) it is relatively easy to implement a preprocessor.

3.1 The Problem:

The input to the preprocessor is a Ratfor source program and the output is a corresponding Fortran program. The preprocessor should either generate Fortran code or simply copy the incoming text after suitable reformatting. The decision is made by examining the first token (not the label) of a Ratfor statement. If it is one of the keywords then translation to an appropriate Fortran code takes place. And if it is of type <u>other</u> (Ref. 2.1), then the statement is outputted without any change.

3.2 Data Structures:

The following are the important data structures used by the preprocessor.

3.2.1 Input File:

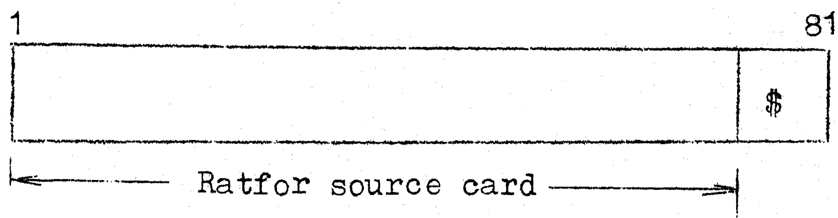
The Ratfor source program is read from the input file.

3.2.2 Output File:

The translated Fortran program is kept on the output file.

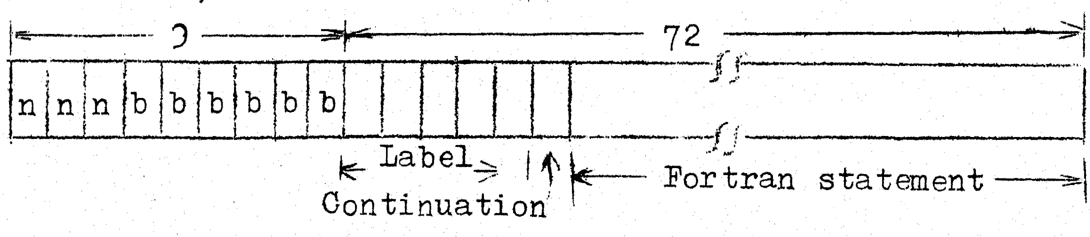
3.2.3 In-Card Buffer-GBUF:

Each card of the Ratfor source program is stored in this buffer in the card image form. The buffer size is 81 characters. The last element of the buffer (i.e. the 81st one) is always a special marker, e.g., a dollar sign, $, which denotes the end of the card.



3.2.4 Out-Card Buffer-OUTBUF:

This buffer contains one Fortran statement, which is obtained after the translation of the source program by the preprocessor. The buffer size is 81 characters. The Fortran statement is stored in the latter 72 fields. The instruction sequence number (ISN), in octal, followed by a few blanks, is filled in the initial fields.



3.2.5 Push Back Stack-BUF:

Any character read in excess from the in-card buffer is pushed back into this stack. This is a simple LIFO stack.

3.2.6 Token-LEXSTR:

The Ratfor source statement is divided into tokens during the lexical analysis. A token can be an alphanumeric string, a quoted string, a single non-alphanumeric character, or an accepted combination of non-alphanumeric characters. LEXSTR, at any given time contains the latest token obtained. The maximum token size allowed is equal to the length of the array LEXSTR.

3.2.7 Terminal Table-TABLE:

All the terminals, i.e., the keywords and the reserved words, are stored in this table. Each character of the terminal goes into one element of the table. The corresponding internal definition of each terminal is also contained in the table, which is organized as follows:

name EOS definition EOS name EOS ...

EOS is a special marker used to separate the names and the definitions, e.g.,

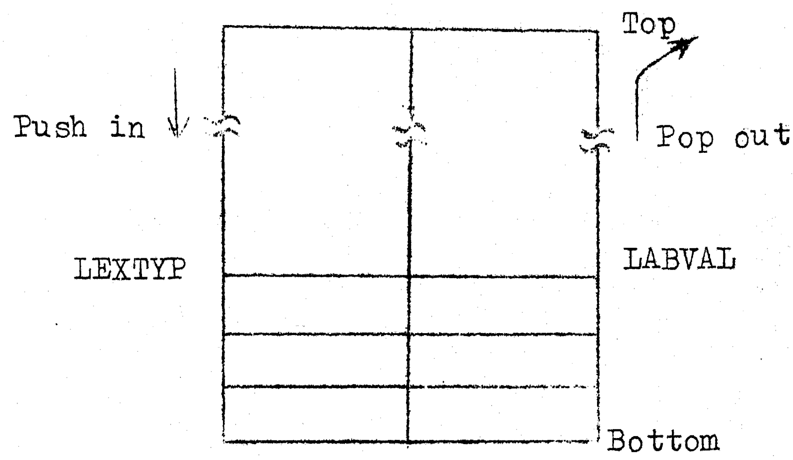
I	F	-2	-10	-2	E	L	S	E	-2	-11	-2	

3.2.8 Identifier Table-IDNTBL:

All those Ratfor identifiers having more than six characters are stored in this table. Consecutive identifiers are separated by the special marker, EOS.

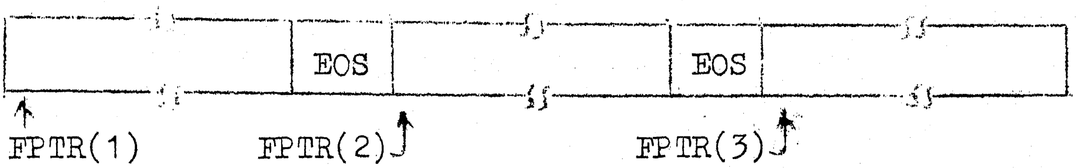
3.2.9 Parser Stacks-LEXTYP and LABVAL:

The definition of the keyword with which a given Ratfor statement begins is stored in stack LEXTYP. The label generated, if any, is entered into the parallel LABVAL stack. These two stacks are very useful in generating the appropriate code in the case of nested control structures. One or more entries at the top of these two stacks are simultaneously popped out when the end of a statement is reached.



3.2.10 Reinitialize Stack-FORBUF:

The reinitialize of a <u>for</u> statement is stored in this stack. This stack lends its importance when dealing with nested <u>for</u> statements, since the reinitialize s that are stored in the stack are outputted in the reverse order. An array pointer, FPTR, stores the beginning of each reinitialize in the stack.



3.3 <u>The Algorithm</u>:

At first, all the relevant data structures, pointers, etc., are initialized by calling the INIT routine. Then the major controlling routine PARSE is called. This routine analyzes (parses) the grammatical structure of the input and takes the necessary action. The PARSE routine can determine the grammatical structure of a Ratfor statement only when the first token (not the label) of the statement is classified. The function GETTOK, as the name suggests, gets a Ratfor token. It also classifies the token as a keyword, identifier, literal, etc. In principle, associated with each rule of the grammar (Ref. 2.1) is a semantic action. In this context, the semantic actions are simply reformatting the incoming text and/or generating the Fortran code.

The PARSE routine also stores information about the tokens that start a valid Ratfor statement, so that the correct terminating code can be produced when the end of the statement is reached. The routine UNSTAK discards the information on which necessary actions have been taken.

When a full line of Fortran statement is obtained, it is kept in the output file. And when the PARSE routine encounters the end-of-file marker (EOF) it returns control to the main program which then outputs the complete translated Fortran program.

3.4 Gudielines for Code Translation:

The basic guidelines by which the various Ratfor statements are translated into an equivalent Fortran code are given below.

3.4.1 IF Statement:

```
Ratfor:          if( <condition> )
                      <statement>

Fortran:         IF (.NOT.( <condition> ))GO TO L
                      <statement>
             L   CONTINUE
```

Let us now take an if-else statement.

```
Ratfor:          if( <condition> )
                      <statement1>
                 else
                      <statement2>

Fortran:         IF(.NOT.( <condition> ))GO TO L
                      <statement1>
                      GO TO L+1
             L   CONTINUE
                 <statement2>
             L+1 CONTINUE
```

When an *if* statement is seen, two consecutive labels L and L+1 are generated. If there is no *else* following an *if*, then the label L+1, though already generated, is not used.

3.4.2 WHILE Statement:

```
Ratfor:          while( <condition> )
                           <statement>

Fortran:         CONTINUE
            L    IF(.NOT.( <condition> ))GO TO L+1
                           <statement>
                            GO TO L
            L+1  CONTINUE
```

The unlabeled CONTINUE before the IF statement takes care of the case when the *while* statement has a label. Then the *while* label is kept in the statement number field of the above CONTINUE.

3.4.3 FOR Statement:

```
Ratfor:          for(<initialize>;<condition>;<reinitialize>
                           <statement>

Fortran:         CONTINUE
                 <initialize>
            L    IF(.NOT.( <condition> ))GO TO L+2
                           <statement>
            L+1            <reinitialize>
                            GO TO L
            L+2  CONTINUE
```

Here three consecutive labels are generated.

3.4.4 REPEAT-UNTIL Statement:

```
Ratfor:        repeat
                        <statement>
               until( <condition> )

Fortran:       CONTINUE
          L    CONTINUE
               <statement>
          L+1  IF(.NOT.( <condition> )) GO TO L
          L+2  CONTINUE
```

An infinite loop occurs when the until part is not present. This is obtained in the Fortran code by replacing the IF(.NOT.(<condition>)) GO TO L by just GO TO L.

3.4.5 DO Statement:

```
Ratfor:        do <limits> <statement>

Fortran:       DO L <limits>
                        <statement>
          L    CONTINUE
          L+1  CONTINUE
```

The do statement in Ratfor is essentially same as the Fortran DO statement with the label, L omitted.

3.4.6 BREAK Statement:

```
Ratfor:        break

Fortran:       GO TO L+1,  when the loop that immediately
                           encloses the break is a while
                           or do

               GO TO L+2   in case of for or repeat
```

3.4.7 NEXT Statement:

Ratfor:	next	
Fortran:	GO TO L	in case of while, infinite repeat or do
	GO TO L+1	in case of for or repeat-until

3.4.8 CHARACTER Declaration:

A character declaration in Ratfor is simply converted into a Fortran INTEGER declaration.

3.5 PARSE and GETTOK Routines:

PARSE and GETTOK are the two major routines. A clear description of these two routines contributes a lot towards the design and understanding of the rest of the preprocessor code.

3.5.1 The PARSE Routine:

Data bases: LEXSTR, LEXTYP, LABVAL

PARSE routine ensures that the code generation operations are done at the right time with the right values. Initially, the first token of a Ratfor source statement is examined. If it happens to be one of the following keywords; if, else, while, do, for, repeat, until and begin, then the corresponding definition is pushed onto the stack LEXTYP. The code generation routine for that keyword is called if there is one. The routines for keywords, if, while, do, for and repeat generate and return a unique label, LAB, which is placed on the parallel stack of label values, LABVAL.

And if the first token is keyword <u>character</u>, then the routine for converting this into a Fortran INTEGER declaration is called. If it is of type digits , then a routine is called which checks the label for possible conflict with the generated labels, and the label is entered into the statement number field of the out-card buffer, OUTBUF.

When the first token is one of the following: type other , <u>break</u>, <u>next</u>, <u>close</u> and semicolon (;), it denotes the end of the statement corresponding to the top of the stack, LEXTYP. The routine UNSTAK is called which may pop one or more entries off the stack as the end of the statement is reached. UNSTAK is turn calls the appropriate routine, if any, that produces the terminating code. The PARSE routine returns control to the main program when it sees the end-of-file marker. This routine also ignores blank lines in the input Ratfor source program.

The following error messages can be reported by the PARSE routine: illegal <u>else</u>, illegal <u>until</u>, stack overflow, illegal <u>close</u> and unexpected end-of-file marker.

3.5.2 The GETTOK Routine:

Data base: LEXSTR.

The function subprogram GETTOK separates the input Ratfor statement into tokens. Comments are ignored. Similarly blanks that separate tokens are also ignored. GETTOK gets a character through NGETC, which obtains one

character at a time from the input including those that are pushed back. PUTBAK pushes back one character onto the input. TYPE determines whether a particular character is a letter or a digit. Two flags SEMCOL and PREV are set to 1 and 4 respectively, whenever a semicolon (.,) or an end-of-line marker ($) is seen, by GETTOK.

When the token obtained is analphanumeric string, then ALLDIG determines if it is a literal (containing only digits), or a possible identifier, or an improper identifier (i.e. an alphanumeric string starting with a digit). When it is a possible identifier then the routine TRMNAL is called. This looks up the terminal table, TABLE, and if the token is found then returns the corresponding definition. If the token is not a terminal symbol, then it is, obviously, an identifier. If the identifier has more than six characters then the routine IDNTFR is called. The long identifier is truncated into six characters. But in order to distinguish between different long identifiers starting with the same first six characters, a separate identifier of length six is generated for each long identifier. This is done by converting the index, an integer number returned by the routine IDNTFR into a character string and appending it after the first few characters of the long identifier. Thus a new six character identifier is produced. This imposes a restriction on the Ratfor identifiers. Any six character identifier should differ from any other longer

identifier in the first four characters. GETTOK increments a line counter for each Ratfor source line that is read.

In summary, the function GETTOK returns the following values, besides the token stored in LEXSTR.

GETTOK	= 0	, if the token is a literal
	= 1	, if the token is an identifier
	= 2	, if the token is a special character
	= 3	, if the token is a possible improper identifier
	= definition	, if the token is a terminal symbol
	= -1	, if the token is the end-of-file marker, EOF
PREV	= 4	, if the token is the end-of-line marker ($)
SEMCOL	= 1	, if the token is a semicolon (.,)

The following error messages can be reported by GETTOK: token too long, missing quote, illegal character, and possible improper identifier.

It is not difficult to understand the significance and the operational details of the other routines (see Appendix D).

3.6 Integrating the Preprocessor with the IBM 7044/1401 System:

In order to execute any given Ratfor program it is sufficient to execute the translated Fortran program, produced

by the preprocessor. The Fortran program can be executed by running it on any Fortran compiler.

There are two Fortran compilers available on the resident IBM 7044/1401 system. One is the $IBFTC compiler under the $IBJOB processor and the other is the in-core WATFOR compiler under the $EXECUTE monitor.

During the preprocessing stage some errors are detected in the Ratfor source program and reported. But these error diagnostics are not exhaustive for the simple reason that in some cases parts of the Ratfor source program are simply reformatted and copied without testing against possible errors. These errors will, of course, be pointed out by the Fortran compiler. A cross reference table between the sequence number of Ratfor statements and the corresponding Fortran statements is made available by the preprocessor. It is easy to trace back any errors, pointed out by the Fortran compiler, to the original Ratfor source program, provided one can a priori know the instruction sequence number (ISN) of the translated Fortran program.

The $IBFTC compiler does not give the ISN's in sequence. It is difficult to know a priori the ISN of any given Fortran statement if the $IBFTC compiler is used. The WATFOR compiler gives the ISNs (in octal) strictly in sequence. And also WATFOR error diagnostics are very good. Thus, the WATFOR compiler is better suited to facilitate trace-back of error diagnostics.

The preprocessor itself is kept under $EXECUTE monitor. The preprocessor reads its input, i.e., the Ratfor source program, from the system input tape (unit no.5), and the output, i.e., the translated Fortran program is written on a scratch tape, say tape 1.

After the preprocessing is done control is transferred to the WATFER compiler, a slightly modified version of WATFOR compiler. The WATFER compiler is loaded into the core over-writing the preprocessor. The only difference between WATFOR and WATFER compilers is that the I/O routines of the WATFOR compiler have to be changed such that it reads input <u>not</u> from the system input tape (unit no.5), but from the scratch tape, i.e., tape 1. This WATFER compiler is also kept under the $EXECUTE monitor. The output of the WATFER compiler, that is the actual output to be obtained after the execution of the Ratfor source program, is however, written on the system output tape (unit no.6).

CHAPTER 4

CONCLUSIONS

The pros and cons of implementing a preprocessor for a language like Ratfor are described. Later, the areas of further work are discussed.

4.1 Pros and Cons [7]

The positive aspects are that

1) the preprocessor attempts to correct the paucity of control structures in Fortran,

2) it is simple to implement a preprocessor,

3) adoption to Ratfor is not difficult because it is close to Fortran.

The following are the less positive factors with regard to the translated Fortran program:

1) It contains some extraneous CONTINUE's, double jumps, and inversions (.NOT.'s). These could be removed by the preprocessor.

2) It could be longer than a Fortran program that is hand-written.

These two disadvantages increase the overhead on compilation. But as the designers of the Ratfor language feel, one might need fewer compilations, because the program works sooner [8].

4.2 Future Work

A macroprocessor can also be developed. One way of designing a macroprocessor is by suitably modifying the PARSER

routine of the preprocessor. Another way of doing this is to develop a macroprocessor that acts as a front-end to the preprocessor. The former method involves lesser overhead, while it is easy to design and understand a full-fledged macroprocessor if the latter technique is adopted.

An execution profile of the Ratfor source program reveals that much time is spent in table look-up. Moreover, when the macroprocessor is also added, the linear search method that is employed becomes inefficient because of higher execution time. The running time can be reduced by adopting either binary search or random entry search (hashing) [4].

When a larger character set is available, one could include all the original Ratfor symbols (e.g., #, ;, < , >, | , &, etc.). It is a simple task to implement these features in the preprocessor that is presently designed.

It would be worthwhile if the various preprocessors that implement structured Fortran are standardized.

In the end Ratfor is just a tool with good points as well as limitations. Writing programs in Ratfor does not necessarily mean one is doing structured programming, nor does it completely solve the problem of large software production [15]. Nevertheless, its merits are numerous and its careful application should prove beneficial.

BIBLIOGRAPHY

1. D.E. Boddy, "Structured Fortran - with or without a preprocessor", SIGPLAN Notices, Vol.12, No.4, pp.34-35, Apr. 1977.

2. W. Brainerd, "Block IF Proposal", FOR-WORD, Vol.2, No.2, Ap Apr. 1976.

3. W. Brainerd, "A proposal for a Fortran loop construct", SIGPLAN Notices, Vol.12, No.12, pp. 60-67, Dec.1977.

4. D. Comer, "MOUSE4: An improved implementation of the Ratfor preprocessor", Software - Practice and Experience, Vol.8, No.1, pp. 34-40, Jan.-Feb. 1978.

5. Draft Proposed ANS Fortran, SIGPLAN Notices, Vol.11, No.3, Mar. 1976.

6. R.A. Fraley, "On replacing Fortran", SIGPLAN Notices, Vol.12, No.9, pp. 130-132, Sept. 1977.

7. E. Horowitz, "FORTRAN: Can it be structured -- Should it be?", Computer, Vol.8, No.6, pp. 30-37, June 1975.

8. B.W. Kernighan and P.J. Plauger, Software Tools, Reading, MA: Addison-Wesley, 1976.

9. D. Knuth, "Structured programming with goto statements", Comput. Surveys, Vol.6, pp. 261-302, Dec. 1974.

10. L.P. Meissner, "Proposed control structures for extended Fortran", SIGPLAN Notices, Vol.11, No.1, pp. 16-21, Jan.1976.

11. L.P. Meissner, "Fortran 77", SIGPLAN Notices, Vol.12, No.1, pp. 93-94, Jan. 1977.

12. D.J. Reifer, "The structured Fortran dilemma", SIGPLAN Notices, Vol.11, No.2, pp.30-32, Feb. 1976.

13. D. Salomon, "A design for Fortran to facilitate structured programming", SIGPLAN Notices, Vol.12, No.1, pp. 95-100, Jan.1977.

14. S. Shinozawa, et al., "Pseudo languages and their preprocessors", Information Processing, pp. 583-587, 1977.

15. L.Stucki, "Automated tools for assisting software development", Practical Strategies for Developing Large Software Systems, edited by E.Horowitz, Reading, MA:Addison-Wesley, 1975.

APPENDIX A

TERMINAL TABLE

The terminal table contains the Ratfor keywords and the reserved words and also their internal definition.

Name	Definition
IF	-10
ELSE	-11
BEGIN	-12
CLOSE	-13
FOR	-14
WHILE	-16
REPEAT	-17
UNTIL	-18
BREAK	-19
CHARACTER	-20
DO	-29
NEXT	-30
INTEGER	-21
LOGICAL	-22
SUBROUTINE	-33
FUNCTION	-45
DOUBLEPRECISION	-46
DIMENSION	-47
COMPLEX	-48
EXTERNAL	-49

APPENDIX B

BRIEF DESCRIPTIONS OF ALL THE ROUTINES

1. INIT - initialize all the relevant data structures and fill in the terminal table

2. PARSE - the major controlling routine that parses the Ratfor source program

3. UNSTAK - unstack at the end of statement

4. GETTOK - get a Ratfor token and classify it accordingly

5. TRMNAL - Locate the keyword/reserved word, and extract definition

6. IDNTFR - stores all the identifiers having more than six characters in the identifier table and returns their respective indices

7. SPCIAL - checks if the given special character is a valid one or not

8. ALLDIG - tests if the given token is an identifier, a literal, or a possible improper identifier

9. TYPE - tests if the given character is a letter or a digit

10. NGETC - get a (possibly pushed back) character

11. PUTBAK - push character back into push back buffer

12. GETC - get characters from the in-card buffer

13. PBSTR - push string back onto input

14. LEXPB - push the latest token obtained into push back buffer

15. TRNSFR - transfers the token or the <u>for</u> reinitialize into an intermediate buffer

16. IFCODE - generate initial code for <u>if</u>

17. DOCODE - generate code for beginning of <u>do</u>

18. WHILEC - generate code for beginning of <u>while</u>

19. FORCOD - generate code for beginning of <u>for</u>

20. REPCOD - generate code for beginning of <u>repeat</u>

21. ELSEIF - generate code for end of <u>if</u> before <u>else</u>

22. REPUNT - generate code for end of <u>repeat</u> before <u>until</u>

23. CHARAC - converts a Ratfor character declaration into a Fortran INTEGER declaration

24. LABELC - output statement number

25. BRKNXT - generate code for <u>break</u> and <u>next</u>

26. OTHERC - output ordinary Fortran statement

27. ENDFOR - generate code for end of <u>for</u>

28. WHILES - generate code for beginning of <u>while</u>

29. DOSTAT - generate code for end of <u>do</u> statement

30. IFGO - generate "IF(.NOT.(<condition>))GO TO L"

31. LABGEN - generate n consecutive labels, return the first one

32. BALPAR - copy a string with balanced parentheses

33. EATUP - process rest of statements; interpret continuations

34. ITOC - convert an integer number into a character string

35. CTOI - convert a character string containing digits into an integer number

36. OCTAL - converts a decimal number into an octal number

37. OUTRAT - outputs a line of Ratfor source program with the line number in octal

38. LEXOUT - puts the latest token in the output buffer

39. OUTTAB - get past column 6

40. OUTNUM - output decimal number

41. OUTGO - output "GO TO N"

42. OUTCON - output "N CONTINUE"

43. OUTDON - keep a full line of translated Fortran program in the output file

44. OUTSTR - output string

45. OUTCH - putsone character into the output buffer

46. ERRMES - report the relevant error message.

APPENDIX C

LIST OF ERROR MESSAGES REPORTED BY THE PREPROCESSOR

1. TOKEN TOO LONG
2. POSSIBLE IMPROPER IDENTIFIER
3. ILLEGAL CHARACTER
4. ILLEGAL ELSE
5. ILLEGAL UNTIL
6. STACK OVERFLOW IN PARSER
7. ILLEGAL CLOSE
8. UNEXPECTED END-OF-FILE MARKER
9. POSSIBLE LABEL CONFLICT
10. MISSING LEFT PARENTHESES
11. UNEXPECTED SPECIAL CHARACTER
12. MISSING CONDITION PART IN FOR STATEMENT
13. MISSING RIGHT PARENTHESIS
14. ILLEGAL NEXT
15. ILLEGAL BREAK
16. UNEXPECTED BEGIN
17. UNBALANCED PARENTHESES
18. TOO MANY CHARACTERS PUSHED BACK ONTO INPUT
19. IDENTIFIER TABLE OVERFLOW
20. MISSING QUOTE
21. DEGREE OF FOR LOOP NESTING EXCEEDS THE LIMIT SPECIFIED
22. UNEXPECTED SEMICOLON
23. STATEMENT NUMBER TOO LARGE.

APPENDIX D

LISTING OF THE RATFOR PREPROCESSOR PROGRAM

AND

SAMPLE OUTPUTS FROM THE PREPROCESSOR

The flow of control in the Ratfor preprocessor can be readily understood by the subroutine tree given in the next page. MAIN denotes the main program and all output routines are removed, since they contribute no complexity.

```
C
C         ****    RATFOR PREPROCESSOR.   ****
C         THIS PREPROCESSOR ACCEPTS A RATFOR SOURCE PROGRAM AS TH INPUT
C         AND GENERATES AN EQUIVALENT FORTRAN PROGRAM AS THE OUTPUT
C         THE ROUTINE 'PARSE' CALLED BY THE MAIN PROGRAM INVOKES THE
C         VARIOUS CODE GENERATION ROUTINES.  THE TRANSLATED FORTRAN PROGRAM
C         SO OBTAINED, IS WRITTEN ON TAPE 1.   'PARSE' RETURNS CONTROL TO THE
C         MAIN PROGRAM WHEN AN END-OF-FILE IS ENCOUNTERED.  THE MAIN
C         PROGRAM THEN OUTPUTS THE FORTRAN PROGRAM AND STOPS.
C
C
      INTEGER OUTBUF(81),OUTP,EOF
      COMMON /L8/ OUTP,OUTBUF
      DATA EOF/1H$/
C                                                                      RAD00034
      CALL INIT                                                        RAD00036
      WRITE(6,104)
      WRITE(6,100)
      REWIND 1
      CALL PARSE
      WRITE(6,101)
      OUTBUF(1)=EOF
      WRITE(1,102)  (OUTBUF(J),J=1,81)
      REWIND 1
   10 READ(1,102)  (OUTBUF(J),J=1,81)
      IF(OUTBUF(1) .EQ. EOF) STOP
        WRITE(6,103)  (OUTBUF(J),J=1,81)
        GO TO 10
  100 FORMAT(25X,*ISN*,35X,*RATFOR SOURCE PROGRAM*,///)
  101 FORMAT(25X,*ISN*,29X,*TRANSLATED FORTRAN PROGRAM*,///)
  102 FORMAT(81A1)
  103 FORMAT(25X,81A1)
  104 FORMAT(1H1)
      END
C
C
      SUBROUTINE INIT
C
C     THIS SUBROUTINE INITIALIZES ALL THE RELEVANT VARIABLES
C
      INTEGER LETTER(26),DIGITS(10),TABLE(300),NAMPTR(40),LEXSTR(30)
      INTEGER BLANK,TOKSIZ,PREV,SEMCOL
      INTEGER OUTBUF(81),LEXTYP(50),LABVAL(50),OUTP,SP,TOKEN
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L2/ LETTER,DIGITS
      COMMON /L4/ TABLE,NAMPTR,MAXPTR,MAXTBL
      COMMON /L5/ LINECT,SEMCOL,PREV
      COMMON /L7/ SP,LEXTYP,LABVAL,TOKEN
      COMMON /L8/ OUTP,OUTBUF
      COMMON /L11/ LABEL
      DATA BLANK/1H /
C
```

```
      READ(5,100) (LETTER(I),I=1,26),(DIGITS(I),I=1,10)
 100  FORMAT(36A1)
      READ(5,101) MAXTBL,MAXPTR
 101  FORMAT(2I3)
      READ(5,102) (TABLE(I),I=1,79)
 102  FORMAT(29X,2A1,9X,A2,2X,I3,2X,A2,/,29X,4A1,7X,A2,2X,I3,2X,A2,/,29
     1 X,5A1,6X,A2,2X,I3,2X,A2,/,29X,5A1,6X,A2,2X,I3,2X,A2,/,29
     1 X,3A1,8X,A2,2X,I3,2X,A2,/,29X,5A1,6X,A2,2X,I3,2X,A2,/,29
     1 X,6A1,5X,A2,2X,I3,2X,A2,/,29X,5A1,6X,A2,2X,I3,2X,A2,/,29
     1 X,5A1,6X,A2,2X,I3,2X,A2,/,29X,9A1,2X,A2,2X,I3,2X,A2)
      READ(5,222) (TABLE(I),I=80,125)
 222  FORMAT(29
     1 X,7A1,4X,A2,2X,I3,2X,A2,/,29X,7A1,4X,A2,2X,I3,2X,A2,/,29
     1 X,6A1,5X,A2,2X,I3,2X,A2,/,29X,7A1,4X,A2,2X,I3,2X,A2,/,29
     1 X,4A1,7X,A2,2X,I3,2X,A2)
      READ(5,121) (TABLE(I), I=126,196)
 121  FORMAT(29
     1 X,5A1,6X,A2,2X,I3,2X,A2,/,29X,6A1,5X,A2,2X,I3,2X,A2,/,29
     1 X,4A1,7X,A2,2X,I3,2X,A2,/,29X,2A1,9X,A2,2X,I3,2X,A2,/,29
     1 X,4A1,7X,A2,2X,I3,2X,A2,/,29X,6A1,5X,A2,2X,I3,2X,A2,/,29
     1 X,4A1,7X,A2,2X,I3,2X,A2,/,29X,10A1,X,A2,2X,I3,2X,A2,/,29
     1 X,3A1,8X,A2,2X,I3,2X,A2)
      READ(5,223) (TABLE(I),I=197,260)
 223  FORMAT(                        7(29X,2A1,9X,A2,2X,I3,2X,A2,/),29
     1 X,3A1,8X,A2,2X,I3,2X,A2,/,29X,2A1,9X,A2,2X,I3,2X,A2,/,29X,4A1,7X,
     1  A2,2X,I3,2X,A2,/,29X,8A1,3X,A2,2X,I3,2X,A2)
      READ(5,104) (NAMPTR(I), I=1,MAXPTR)
 104  FORMAT(20I4)
      LINECT=1
      SEMCOL=0
      LINECT=1
      PREV=-2
      OUTP=0
      SP=1
      LABEL=30000
      DO 10 I=1,81
 10     OUTBUF(I)=BLANK
      DO 11 I = 1,30
 11   LEXSTR(I)=BLANK
      DO 12 I = 1,50
      LEXTYP(I)=0
 12   LABVAL(I)=0
      LEXTYP(1)=-1
      RETURN
      END
C
C
      SUBROUTINE PARSE
C
C     THIS ROUTINE PARSES THE INPUT SOURCE PROGRAM AND IN TURN CALLS
C     THE APPROPRIATE CODE GENERATION ROUTINES
```

```
      INTEGER  LEXSTR(30),LABVAL(50),LEXTYP(50),PREV,TOKSIZ,TOKEN
      INTEGER GETTOK,SEMCOL,SP
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L5/ LINECT,SEMCOL,PREV
      COMMON /L7/ SP,LEXTYP,LABVAL,TOKEN
      DATA MAXSTK/50/
 200  TOKEN=GETTOK(JACK)
      IF(TOKEN+1) 201,500,201
 201  IF(PREV.EQ.4 .OR. SEMCOL.EQ.1)  GO TO 200
      IF(TOKEN .NE. (-10)) GO TO 202
          CALL IFCODE(LAB)
          GO TO 212
 202  IF(TOKEN .NE. (-29)) GO TO 203
          CALL DOCODE(LAB)
          GO TO 212
 203  IF(TOKEN .NE. (-16)) GO TO 204
          CALL WHILEC(LAB)
          GO TO 212
 204  IF(TOKEN .NE. (-14)) GO TO 205
          CALL FORCOD(LAB)
          GO TO 212
 205  IF(TOKEN .NE. (-17)) GO TO 206
          CALL REPCOD(LAB)
          GO TO 212
 206  IF(TOKEN .NE. (-11)) GO TO 208
          IF(LEXTYP(SP) .NE. (-10)) GO TO 207
              CALL ELSEIF(LABVAL(SP))
              GO TO 212
C*** ERROR MESSAGE - ILLEGAL ELSE
 207      ISN=4
          CALL ERRMES(ISN)
          GO TO 212
 208  IF(TOKEN .NE. (-18)) GO TO 210
          IF(LEXTYP(SP) .NE. (-17)) GO TO 209
              CALL REPUNT(LABVAL(SP))
              GO TO 212
C*** ERROR MESSAGE - ILLEGAL UNTIL
 209      ISN=5
          CALL ERRMES(ISN)
          GO TO 212
 210  IF(TOKEN .NE. (-20)) GO TO 211
          CALL CHARAC
          GO TO 200                                                    RAD011
 211  IF(TOKEN .NE. 0) GO TO 212
          CALL LABELC
          GO TO 200
 212  IF(TOKEN.NE.(-10) .AND. TOKEN.NE.(-11) .AND. TOKEN.NE.(-12) .AND.
     1   TOKEN.NE.(-14) .AND. TOKEN.NE.(-16) .AND. TOKEN.NE.(-17) .AND.
     1   TOKEN.NE.(-18) .AND. TOKEN.NE.(-29)) GO 00  1
          SP=SP+1
```

```
              IF(SP .LE. MAXSTK) GO TO 213
C*** ERROR MESSAGE - STACK OVERFLOW
                    ISN=6
                    CALL ERRMES(ISN)
  213             LEXTYP(SP)=TOKEN
                  LABVAL(SP)=LAB
                  GO TO 200
  214   IF(TOKEN .NE. (-13)) GO TO 216
            IF(LEXTYP(SP) .NE. (-12)) GO TO 215
              SP=SP-1
              GO TO 218
C*** ERROR MESSAGE - ILLEGAL CLOSE
  215       ISN=7
            CALL ERRMES(ISN)
            GO TO 218
  216   IF(TOKEN .NE. (-19) .AND. TOKEN .NE. (-30)) GO TO 217
            CALL BRKNXT
            GO TO 218
  217   CALL OTHERC
C*** PEEP AT THE NEXT TOKEN
  218   TOKEN=GETTOK(JACK)
        IF(TOKEN+1) 219,500,219
  219   IF(PREV.EQ.4 .OR. SEMCOL.EQ.1) GO TO 218
        CALL LEXPB
        CALL UNSTAK
        GO TO 200
  500   IF(SP .EQ. 1) RETURN
C*** ERROR MESSAGE - UNEXPECTED END-OF-FILE
            ISN=8
            CALL ERRMES(ISN)
            RETURN
        END
C
C
        INTEGER FUNCTION GETTOK(JACK)
        INTEGER LEXSTR(30),Z,EOF,C,BLANK,TYPE,TOKSIZ,SLASH,DOT,STAR
        INTEGER COMMA,PREV,NULINE,ALLDIG,SEMCOL,SPDEX,TRMDEX,DEFN
        INTEGER STR(11),QUOTE
        COMMON /L1/ LEXSTR,TOKSIZ
        COMMON /L5/ LINECT,SEMCOL,PREV
        DATA EOF/2H*$/,BLANK/1H /,SLASH/1H//,DOT/1H./,QUOTE/1H'/
        DATA NULINE/1H$/,STAR/1H*/,COMMA/1H,/
        DATA EOF/2H*$/,BLANK/1H /,SLASH/1H//,DOT/1H./
C
   22   SEMCOL=0
    1   Z=NGETC(C)
        IF(Z .EQ. EOF) GO TO 15
            IF(C .EQ. BLANK) GO TO 1
                CALL PUTBAK(C)
                I=1
    2           IF(I .GE. 22) GO TO 3
```

```
                  Z=NGETC(LEXSTR(I))
                  IF(TYPE(Z) .EQ. 2) GO TO 3
                      I=I+1
                      GO TO 2
    3         IF(I .LT. 22) GO TO 5
                  TOKSIZ=20
                  I=I-1
C*** ERROR MESSAGE - TOKEN TOO LONG
                  ISN=1
                  CALL ERRMES(ISN)
                  CALL PUTBAK(LEXSTR(I))
    4             Z=NGETC(C)
                  IF(Z .EQ. EOF) GO TO 15
                      IF(TYPE(C) .NE. 2) GO TO 4
                      CALL PUTBAK(C)
                      GO TO 14
    5         IF(I .GT. 1) GO TO 13
                  Z=LEXSTR(1)
                  IF(Z .EQ. SLASH) GO TO 6
                  IF(Z .EQ. DOT) GO TO 7
                  IF(Z .NE. NULINE) GO TO 16
                      GETTOK=2
                      TOKSIZ=1
                      PREV=4
                      LINECT=LINECT+1
                      RETURN
   16             IF(Z .NE. QUOTE) GO TO 25
                      I=1
   24                 I=I+1
                      IF(NGETC(LEXSTR(I)) .EQ. QUOTE) GO 00  5
                      IF(LEXSTR(I).NE.QUOTE .AND. I.LT.29) GOTO24
                          ISN=20
                          CALL ERRMES(ISN)
                          LEXSTR(I)=QUOTE
                          CALL PUTBAK(NULINE)
   25             CALL SPCIAL(SPDEX)
                  IF(SPDEX+1) 18,17,18
C*** ERROR MESSAGE - ILLEGAL CHARACTER
   17                 ISN=3
                      CALL ERRMES(ISN)
                      GO TO 22
C*** TOKEN IS A SPECIAL CHARACTER
   18             GETTOK=2
                  TOKSIZ=1
                  PREV=GETTOK
                  RETURN
    6             IF(NGETC(LEXSTR(2)) .EQ. STAR) GO TO 10
                      GO TO 8
    7             IF(NGETC(LEXSTR(2)) .EQ. COMMA) GO TO 9
    8                 CALL PUTBAK(LEXSTR(2))
                      GETTOK=2
```

```
                              TOKSIZ=1
                              PREV=GETTOK
                              RETURN
C*** TOKEN IS A SEMICOLON
    9                    GETTOK=-43
                         TOKSIZ=2
                         PREV=GETTOK
                         SEMCOL=1
                         RETURN
C*** STRIP THE COMMENTS
   10                    IF(PREV .EQ. (-2)) GO TO 12
                              IF(PREV .EQ. 4) GO TO 12
   11                              CONTINUE
                                   IF(NGETC(LEXSTR(1)) .NE. NULINE) GO TO 11
                                        GETTOK=2
                                        TOKSIZ=1
                                        PREV=4
                                        LINECT=LINECT+1
                                        RETURN
   12                    CONTINUE
                         IF(NGETC(LEXSTR(1)) .NE. NULINE) GO TO 12
                              GO TO 22
   13               CALL PUTBAK(LEXSTR(I))
                    TOKSIZ=I-1
   14               GETTOK=ALLDIG(JILL)
                    PREV=GETTOK
                    IF(GETTOK .NE. 3) GO TO 19
C*** ERROR MESSAGE - POSSIBLE IMPROPER IDENTIFIER
                         ISN=2
                         CALL ERRMES(ISN)
                         GO TO 21
   19               IF(GETTOK .NE. 1) RETURN
                         CALL TRMNAL(TRMDEX,DEFN)
                         IF(TRMDEX+1) 20,21,20
C*** TOKEN IS A TERMINAL SYMBOL
   20                         GETTOK=DEFN
                              RETURN
   21                    GETTOK=1
                         IF(TOKSIZ .LE. 6) RETURN
                              CALL IDNTFR(IDNDEX)
                              LEN=ITOC(IDNDEX,STR)
                              IF(LEN .NE. 1) GO TO 23
                                   TOKSIZ=6
                                   LEXSTR(6)=STR(1)
                                   RETURN
   23                         TOKSIZ=6
                              LEXSTR(5)=STR(1)
                              LEXSTR(6)=STR(2)
                              RETURN
C*** END-OF-FILE ENCOUNTERED
   15   GETTOK=-1
```

```
      RETURN
      END
C
C
      SUBROUTINE UNSTAK
      INTEGER LEXTYP(50),LABVAL(50),SP,TOKEN
      COMMON /L7/ SP,LEXTYP,LABVAL,TOKEN
C
C     THIS SUBROUTINE TAKES CARE OF THE END OF A RATFOR STATEMENT
C     AND, IF NECESSARY, GENERATES THE FINISHING CODE
C
      SP=SP+1
    1 SP=SP-1
      IF(SP .LE. 1) RETURN
      IF(LEXTYP(SP) .EQ. (-12)) RETURN
      IF(LEXTYP(SP) .EQ. (-10) .AND. TOKEN .EQ. (-11)) RETURN
         IF(LEXTYP(SP) .NE. (-10)) GO TO 2
            CALL OUTCON(LABVAL(SP))
            GO TO 1
    2 IF(LEXTYP(SP) .NE. (-11)) GO TO 3
         IF(SP .GT. 2) SP=SP-1
         JAYA=LABVAL(SP)+1
         CALL OUTCON(JAYA)
         GO TO 1
    3 IF(LEXTYP(SP) .EQ. (-17) .AND. TOKEN .EQ. (-18)) RETURN
         IF(LEXTYP(SP) .NE. (-17)) GO TO 4
            CALL OUTGO(LABVAL(SP))
            JAYA=LABVAL(SP)+1
            CALL OUTCON(JAYA)
            GO TO 1
    4 IF(LEXTYP(SP) .NE. (-29)) GOTO 5
         CALL DOSTAT(LABVAL(SP))
         GO TO 1
    5 IF(LEXTYP(SP) .NE. (-14)) GO TO 6
         CALL ENDFOR(LABVAL(SP))
         GO TO 1
    6 IF(LEXTYP(SP) .EQ. (-16)) CALL WHILES(LABVAL(SP))
      GO TO 1
      END
C
C
      SUBROUTINE BRKNXT
C
C     GENERATE CODE FOR BREAK AND NEXT STATEMENTS
C
      INTEGER LEXTYP(50),LABVAL(50),SP,TOKEN
      COMMON /L7/ SP,LEXTYP,LABVAL,TOKEN
C
      I=SP+1
    1 I=I-1
      IF(I .LE. 0) GO TO 5
```

```
          LEX=LEXTYP(I)
          IF(LEX.NE.(-16) .AND. LEX.NE.(-17) .AND. LEX.NE.(-29)) GOTO 2
              JAYA=LABVAL(I)
              GO TO 3
    2     IF(LEX .NE. (-14)) GOTO 1
              JAYA=LABVAL(I)+1
    3         IF(TOKEN .NE. (-19)) GO TO 4
C*** GENERATE CODE FOR BREAK
                  LATA=JAYA+1
                  CALL OUTGO(LATA)
                  RETURN
C*** GENERATE CODE FOR NEXT
    4         CALL OUTGO(JAYA)
              RETURN
    5     IF(TOKEN .EQ. (-19)) GO TO 6
C*** ERROR MESSAGE - ILLEGAL NEXT
          ISN=14
          CALL ERRMES(ISN)
          RETURN
C*** ERROR MESSAGE - ILLEGAL BREAK
    6 ISN=15
      CALL ERRMES(ISN)
      RETURN
      END
C
C
      SUBROUTINE FORCOD(LAB)
C
C     THIS SUBROUTINE GENERATES THE CODE AT THE BEGINNING OF A
C     FOR STATEMENT
C     FLAG1(I) = 1 WHEN THE I'TH FOR LOOP HAS A MISSING CONDITION PART
C     FLAG2(I) = 0 WHEN THE I'TH FOR LOOP HAS NO 'REINITIALIZE' PART
C
      INTEGER FORBUF(50),LEXSTR(30),IFNOT1(9), TOKSIZ,PREV
      INTEGER FLAG1(5),FLAG2(5),FPTR(5),SEMCOL,GETTOK,RPAREN,EOS
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L5/ LINECT,SEMCOL,PREV
      COMMON /L6/ FORBUF,FLAG1,FLAG2,FPTR,K,KNT
      DATA LPAREN/1H(/,RPAREN/1H)/,EOS/2H-2/,FORBUF/50*1H /
      DATA IFNOT1/1HI,1HF,1H(,1H.,1HN,1HO,1HT,1H.,1H(/
      DATA K/1/,KNT/0/,MAX/5/
C
      KNT=KNT+1
      IF(KNT .LE. MAX) GO TO 20
         ISN=21
         CALL ERRMES(ISN)
         RETURN
   20 FPTR(KNT)=K
      DO 21 I = KNT,MAX
        FLAG1(I)=0
   21   FLAG2(I)=1
```

```
      LAB=LABGEN(3)
      CALL OUTCON(0)
      CALL OUTTAB
      INDU=GETTOK(JACK)
      IF(INDU .EQ. 2) GO TO 1
C*** ERROR MESSAGE - LEFT PAREN. MISSING
            ISN=10
            CALL ERRMES(ISN)
            GO TO 3
    1 IF(LEXSTR(1) .EQ. LPAREN) GO TO 2
C*** ERROR MESSAGE - SOME OTHER SPECIAL CHARACTER IN PLACE OF LEFT PAREN.
            ISN=11
            CALL ERRMES(ISN)
    2 INDU=GETTOK(JACK)
    3 IF(SEMCOL .EQ. 1) GOTO 4
C*** COPY THE INITIALIZE PART
   22       CALL LEXOUT
            INDU=GETTOK(JACK)
            IF(SEMCOL .NE. 1) GOTO 22
            CALL OUTDON
    4 INDU=GETTOK(JACK)
      CALL LEXP8
      IF(SEMCOL .NE. 1) GO TO 6
C*** ERROR MESSAGE - MISSING CONDITION PART
            ISN=12
            CALL ERRMES(ISN)
            FLAG1(KNT)=1
    5       INDU=GETTOK(JACK)
C*** SKIP THE REST OF THE LINE
            IF(PREV .NE. 4) GO TO 5
                  RETURN
    6 CALL OUTNUM(LAB)
      CALL OUTTAB
      CALL OUTSTR(IFNOT1,9)
      INDU=GETTOK(JACK)
C*** COLLECT AND COPY THE CONDITION PART
    7 CALL LEXOUT
      INDU=GETTOK(JACK)
      IF(SEMCOL .NE. 1) GO TO 7
            CALL OUTCH(RPAREN)
            CALL OUTCH(RPAREN)
            JAYA=LAB+2
            CALL OUTGO(JAYA)
      INDU=GETTOK(JACK)
      IF(INDU .NE. (-12)) GO TO 12
            CALL LEXP8
            GO TO 13
   12 IF(PREV .NE. 4) GO TO 9
C*** ERROR MESSAGE - MISSING RIGHT PAREN.
   13       ISN=13
            CALL ERRMES(ISN)
```

```
          GO TO 11
    9   IF(LEXSTR(1) .EQ. RPAREN) GO TO 11
C*** KEEP THE REINITIALIZE PART IN A BUFFER
          DO 10 I = 1,TOKSIZ
            FORBUF(K)=LEXSTR(I)
   10       K=K+1
          GO TO 8
C*** IF BUFFER EMPTY, RESET FLAG2
   11   IF(FPTR(KNT) .EQ. K) GO TO 25
          FORBUF(K)=EOS
          K=K+1
          RETURN
   25   FLAG2(KNT)=0
        RETURN
        END
C
C
        SUBROUTINE ENDFOR(LAB)
C*** GENERATE CODE AT THE END OF A FOR STATEMENT
        INTEGER FORBUF(50),LBUF(50),FLAG1(5),FLAG2(5),FPTR(5)
        COMMON /L6/ FORBUF,FLAG1,FLAG2,FPTR,K,KNT
C
        IF(FLAG1(KNT) .EQ. 1 )  GO TO 2
        IF(FLAG2(KNT) .NE. 1)   GOTO 1
C***    OUTPUT THE 'INITIALIZE' PART WITH THE PROPER LABEL
            JAYA=LAB+1
            CALL OUTNUM(JAYA)
            CALL OUTTAB
            CALL TRNSFR(LBUF,LEN,2)
            CALL OUTSTR(LBUF,LEN)
            CALL OUTDON
    1   CALL OUTGO(LAB)
        JAYA=LAB+2
        CALL OUTCON(JAYA)
    2   KNT=KNT-1
        K=K-LEN-1
        RETURN
        END
C
C
        SUBROUTINE IFCODE(LAB)
        LAB=LABGEN(2)
        CALL IFGO(LAB)
        RETURN
        END
C
C
        SUBROUTINE DOCODE(LAB)
C*** GENERATE CODE FOR BEGINNING OF DO STATEMENT
        INTEGER DOSTR(2)
        DATA DOSTR/1HD,1HO/
```

```
C
      CALL OUTTAB
      CALL OUTSTR(DOSTR,2)
      LAB=LABGEN(2)
      CALL OUTNUM(LAB)
      CALL EATUP
      CALL OUTDON
      RETURN
      END
C
C
      SUBROUTINE WHILEC(LAB)
C*** WHILEC - GENERATE CODE FOR BEGINNING OF WHIEL STATEMENT
      CALL OUTCON(0)
      LAB=LABGEN(2)
      CALL OUTNUM(LAB)
      JAYA=LAB+1
      CALL IFGO(JAYA)
      RETURN
      END
C
C
      SUBROUTINE REPCOD(LAB)
C*** GENERATE THE INITIAL CODE FOR REPEAT STATEMENT
      LAB=LABGEN(2)
      CALL OUTCON(0)
      CALL OUTCON(LAB)
      RETURN
      END
C
C
      SUBROUTINE ELSEIF(LAB)
C*** ELSE IF -GENERATE CODE FOR ENDOF IF BEFORE ELSE
      JAYA=LAB+1
      CALL OUTGO(JAYA)
      CALLOUTCON(LAB)
      RETURN
      END
C
C
      SUBROUTINE REPUNT(LAB)
C*** GENERATE CODE FOR AN UNTIL PRECEDED BY A REPEAT
      JAYA=LAB+1
      CALL IFGO(LAB)
      CALL OUTCON(JAYA)
      RETURN
      END
C
C
      SUBROUTINE CHARAC
C*** CONVERT THE CHARACTER INTO AN INTEGER DECLARATION
```

```
      INTEGER INTGER(7)
      DATA INTGER/1HI,1HN,1HT,1HE,1HG,1HE,1HR/
      CALL OUTTAB
      CALL OUTSTR(INTGER,7)
      CALL EATUP
      CALL OUTDON
      RETURN
      END
C
C
      SUBROUTINE LABELC
C*** LABELC - OUTPUT STATEMENT NUMBER
      INTEGER LEXSTR(30),TOKSIZ,LBUF(50)
      COMMON /L1/ LEXSTR,TOKSIZ
C
      IF(TOKSIZ .NE. 5) GO TO 2
         CALL TRNSFR(LBUF,LEN,1)
         CALL CTOI(LBUF,LEN,NUM)
         IF(NUM .LT. 30000) GO TO 1
            ISN=9
            CALL ERRMES(ISN)
    1 CALL OUTNUM(NUM)
      CALL OUTTAB
      RETURN
    2 IF(TOKSIZ .LT. 5) GO TO 1
         ISN=23
         CALL ERRMES(ISN)
         RETURN
      END
C
C
      SUBROUTINE OTHERC
C*** OTHERC- OUTPUT ORDINARY FORTRAN STATEMENT
C
      CALL OUTTAB
      CALL LEXOUT
      CALL EATUP
      CALL OUTDON
      RETURN
      END
C
C
      SUBROUTINE DOSTAT(LAB)
C *** DOSTAT - GENERATE CODE FOR END OF DO STATEMENT
      CALL OUTCON(LAB)
      LAB1=LAB+1
      CALL OUTCON(LAB1)
      RETURN
      END
C
C
```

```
      SUBROUTINE IFGO(LAB)
C ***  IFGO - GENERATE 'IF(.NOT.( ... )) GO TO LAB'
      INTEGER IFNOT(8),RPAREN
      DATA IFNOT/1HI,1HF,1H(,1H.,1HN,1HO,1HT,1H./,RPAREN/1H)/
C
      CALL OUTTAB
      CALL OUTSTR(IFNOT,8)
      CALL BALPAR
      CALL OUTCH(RPAREN)
      CALL OUTGO(LAB)
      RETURN
      END
C
C
      SUBROUTINE WHILES(LAB)
C  ***  WHILES - GENERATE CODE FOR END OF WHILE STATEMENT
      CALL OUTGO(LAB)
      LAB1=LAB+1
      CALL OUTCON(LAB1)
      RETURN
      END
C
C
      INTEGER FUNCTION LABGEN(N)
C*** GENERATE N CONSECUTIVE LABLES AND RETURN THE FORST ONE
      COMMON /L11/ LABEL
      LABGEN=LABEL
      LABEL=LABEL+N
      RETURN
      END
C
C
      SUBROUTINE BALPAR
C*** COPY BALANCED PARENTHESIS STRING
      INTEGER LEXSTR(30),GETTOK,TOKSIZ,SEMCOL,PREV,RPAREN
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L5/ LINECT,SEMCOL,PREV
      DATA LPAREN/1H(/,RPAREN/1H)/
C
      INDU=GETTOK(JACK)
      IF(INDU .EQ. 2) GO TO 7
C*** ERROR MESSAGE - MISSING LEFT PAREN.
          ISN=10
          CALL ERRMES(ISN)
          CALL OUTCH(LPAREN)
          GO TO 8
    7 IF(LEXSTR(1) .EQ. LPAREN) GO TO 1
C*** ERROR MESSAGE - SOME OTHER SPECIAL CHARACTER IN PLACE OF LEFT PAREN.
          ISN=11
          CALL ERRMES(ISN)
          CALL OUTCH(LPAREN)
```

```
      GO TO 8
    1 CALL LEXOUT
    8 NLPAR=1
    2 INDU=GETTOK(JACK)
      IF(INDU.EQ.(-43) .OR. INDU.EQ.(-12) .OR. INDU.EQ.(-13) .OR.
     1 INDU .EQ. (-1)) GO TO 5
      IF( PREV .EQ. 4) GO TO 2
      IF(LEXSTR(1) .NE. LPAREN) GO TO 3
         NLPAR=NLPAR+1
         GO TO 4
    3 IF(LEXSTR(1) .NE. RPAREN) GO TO 4
         NLPAR=NLPAR-1
    4 CALL LEXOUT
      IF(NLPAR .GT. 0) GO TO 2
      GO TO 6
    5 CALL LEXPB
    6 IF(NLPAR .EQ. 0) RETURN
         ISN=17
         CALL ERRMES(ISN)
      RETURN
      END
C
C
      SUBROUTINE EATUP
      INTEGER LEXSTR( 30),TOKSIZ,GETTOK,COMMA,RPAREN,SEMCOL,PREV
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L5/ LINECT,SEMCOL,PREV
      DATA COMMA/1H,/,LPAREN/1H(/,RPAREN/1H)/
C
      NLPAR=0
    1 INDU=GETTOK(JACK)
      IF(SEMCOL.EQ.1 .OR. PREV.EQ.4) GO TO 9
      IF(INDU .NE. (-13)) GO TO 2
         CALL LEXPB
         GO TO 9
    2 IF(INDU .NE. (-1)) GO TO 3
C*** ERROR MESSAGE - UNEXPECTED END-OF-FILE
         ISN=8
         CALL ERRMES(ISN)
         CALL LEXPB
         GO TO 9
    3 IF(INDU .NE. (-12)) GO TO 4
C*** ERROR MESSAGE - UNEXPECTED BEGIN
         ISN=16
         CALL ERRMES(ISN)
         CALL LEXPB
         GO TO 9
C*** CHECK FOR CONTINUATION
    4 IF(LEXSTR(1) .NE. COMMA) GO TO 5
         INDU=GETTOK(JACK)
         IF(PREV .EQ. 4) GO TO 1
```

```
            CALL LEXPB
            CALL OUTCH(COMMA)
            GO TO 8
  5   IF(LEXSTR(1) .NE. LPAREN) GO TO 6
        NLPAR=NLPAR+1
        CALL OUTCH(LPAREN)
        GO TO 8
  6   IF(LEXSTR(1) .NE. RPAREN) GO TO 7
        NLPAR=NLPAR-1
        CALL OUTCH(RPAREN)
        GO TO 8
  7   CALL LEXOUT
  8   IF(NLPAR .GE. 0) GO TO 1
  9   IF(NLPAR .EQ. 0) RETURN
C*** ERROR MESSAGE - UNBALANCED PARENTHESES
        ISN=17
        CALL ERRMES(ISN)
        RETURN
      END
C
C
      SUBROUTINE LEXPB
C*** THE LATEST TOKEN IS PUT  BACK ONTO INPUT
      INTEGER LBUF(50)
      CALL TRNSFR(LBUF,LEN,1)
      CALL PBSTR(LBUF,LEN)
      RETURN
      END
C
C
      SUBROUTINE LEXOUT
C*** THE LATEST TOKEN IS PUT IN  THE OUTPUT BUFFER
      INTEGER LBUF(50)
      CALL TRNSFR(LBUF,LEN,1)
      CALL OUTSTR(LBUF,LEN)
      RETURN
      END
C
C
      SUBROUTINE TRNSFR(LBUF,LEN,NO)
      INTEGER LEXSTR(30),LBUF(50),FORBUF(50)
      INTEGER FLAG1(5),FLAG2(5),FPTR(5),TOKSIZ,EOS
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L6/ FORBUF,FLAG1,FLAG2,FPTR,K,KNT                      RAD0686
      DATA EOS/2H-2/                                                 RAD0686
C
      GO TO(1,2),NO
  1   LEN=TOKSIZ
      DO 10 I = 1,LEN
 10     LBUF(I)=LEXSTR(I)
      RETURN
```

```
    4   NLEN=TOKSIZ+1



    2   J=FPTR(KNT)
        I=1
    3   LBUF(I)=FORBUF(J)
        I=I+1
        J=J+1
        IF(FORBUF(J) .NE. EOS) GO TO 3
        LEN=I-1
        RETURN
        END
C
C
        SUBROUTINE PBSTR(LBUF,LEN)
C*** PUT BACK THE GIVEN STRING
        INTEGER LBUF(50),C
        DO 10 I = 1,LEN
          I1=LEN-I+1
          C=LBUF(I1)
   10     CALL PUTBAK(C)
        RETURN
        END
        SUBROUTINE TRMNAL(TRMDEX,DEFN)
C
C       THIS SUBROUTINE SEARCHES THE TERMINAL TABLE FOR THE A-PHA TOKEN
C
        INTEGER LEXSTR(30),NAMPTR(40),TABLE(300),EOS,DEFN,TRMDEX,TOKSIZ
        COMMON /L1/ LEXSTR,TOKSIZ
        COMMON /L4/ TABLE,NAMPTR,MAXPTR,MAXTBL
        DATA EOS/2H-2/
C
        LEXSTR(TOKSIZ+1) = EOS
        LASTP=1
        I=LASTP-1
    1   I=I+1
        IF(I .GT. MAXPTR) GO TO 4
            J=NAMPTR(I)
            K=1
    2       IF(LEXSTR(K) .NE. TABLE(J) .OR. LEXSTR(K) .EQ. EOS) GO TO 3
                J=J+1
                K=K+1
                GO TO 2
    3       IF(LEXSTR(K) .NE. TABLE(J)) GO TO 1
C*** SEARCH SUCCESSFUL
                DEFN=TABLE(J+1)
                TRMDEX=I
                RETURN
C*** SEARCH UNSUCCESSFUL
    4   TRMDEX=-1
        RETURN
        END
C
C
```

```
    4  NLEN=TOKSIZ+1
       LENGTH=LTBL+NLEN
       IF(LENGTH .LE. MAX) GO TO 5
C*** ERROR MESSAGE - IDENTIFIER TABLE OVERFLOWS
            ISN=19
            CALL ERRMES(ISN)
            RETURN
    5  LPTR=LPTR+1
       IDNPTR(LPTR)=LTBL+1
       DO 6 I = 1,TOKSIZ
       LTBL=LTBL+1
    6  IDNTBL(LTBL)=LEXSTR(I)
       LTBL=LTBL+1
```

```
SUB2C    NOPRNT
SUBROUTINE IDNTFR(IDNDEX)
HE TABLE IS CHECKED FOR IDENTIFIERS THAT ARE LONGER THAN
IX CHARACTERS. IF FOUND, THE CORRESPONDING INDEX IS RETURNED.
LSE, THE IDENTIFIER IS ENTERED INTO THE TABLE.
INTEGER LEXSTR(30),IDNPTR(20),IDNTBL(200),TOKSIZ,EOS
COMMON /L1/ LEXSTR,TOKSIZ
DATA KNT/0/,MAX/200/,EOS/2H-2/,LPTR/0/,IDNTBL/200*1H /,LTBL/0/

KNT=KNT+1
IF(KNT .EQ. 1) GO TO 4
    LEXSTR(TOKSIZ+1)=EOS
    I=LPTR+1
    I=I-1
    IF(I .LE. 0) GO TO 4
        J=IDNPTR(I)
        K=1
        IF(LEXSTR(K).NE.IDNTBL(J) .OR. LEXSTR(K).EQ.EOS) GO TO 3
            J=J+1
            K=K+1
            GO TO 2
        IF(LEXSTR(K) .NE. IDNTBL(J)) GO TO 1
            IDNDEX=I
            RETURN
```

```
C
C        TYPE = 0   IF C IS A DIGIT
C        TYPE = 1   IF C IS A LETTER
C        TYPE = 2   IF C IS NEITHER OF THE ABOVE
C
         INTEGER LETTER(26),DIGITS(10),C
         COMMON /L2/ LETTER,DIGITS
C
         DO 1 I = 1,10
         IF (C .EQ. DIGITS(I)) GO TO 2
    1           CONTINUE
```

```
    4  NLEN=TOKSIZ+1
       LENGTH=LTBL+NLEN
       IF(LENGTH .LE. MAX) GO TO 5
C*** ERROR MESSAGE - IDENTIFIER TABLE OVERFLOWS
          ISN=19
          CALL ERRMES(ISN)
          RETURN
    5  LPTR=LPTR+1
       IDNPTR(LPTR)=LTBL+1
       DO 6 I = 1, TOKSIZ
       LTBL=LTBL+1
    6  IDNTBL(LTBL)=LEXSTR(I)
       LTBL=LTBL+1
       IDNTBL(LTBL)=EOS
       IDNDEX=LPTR
       RETURN
       END
C
C
       SUBROUTINE SPCIAL(SPDEX)
C*** CHECKS IF THE INPUT CHARACTER IS A VALID SPECIAL CHARACTER
       INTEGER LEXSTR(30),SPCHAR(11),SPDEX,TOKSIZ
       COMMON /L1/ LEXSTR, TOKSIZ
       DATA SPCHAR/1H+,1H-,1H*,1H/,1H.,1H,,1H(,1H),1H=,1H',1H$/
C
       DO 10 I = 1,11
       LATA=I
       IF(LEXSTR(1) .EQ. SPCHAR(I)) GO TO 1
   10     CONTINUE
C*** SEARCH UNSUCCESSFUL
          SPDEX=-1
          RETURN
    1  SPDEX=LATA
       RETURN
       END
C
C
       INTEGER FUNCTION TYPE(C)
C
C      TYPE = 0  IF C IS A DIGIT
C      TYPE = 1  IF C IS A LETTER
C      TYPE = 2  IF C IS NEITHER OF THE ABOVE
C
       INTEGER LETTER(26),DIGITS(10),C
       COMMON /L2/ LETTER, DIGITS
C
       DO 1 I = 1,10
       IF (C .EQ. DIGITS(I)) GO TO 2
    1        CONTINUE
```

```
      DO 3 J = 1,26
      IF(C .EQ. LETTER(J)) GO TO 4
    3         CONTINUE
              TYPE=2
              RETURN
    2 TYPE=0
      RETURN
    4 TYPE=1
      RETURN
      END
C
C
      INTEGER FUNCTION ALLDIG(JILL)
C
C     ALLDIG = 0  ***  ALL THE CHARACTERS ARE DIGITS
C
      INTEGER LEXSTR(30),LETTER(26),DIGITS(10),TOKSIZ,EOS
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L2/ LETTER,DIGITS
      DATA EOS/2H-2/
C
      ALLDIG=1
      LEXSTR(TOKSIZ+1) = EOS
      I = 0
    1 I = I + 1
      IF( LEXSTR(I) .EQ. EOS ) GO TO 3
          DO 2 J = 1,10
          IF( LEXSTR(I) .EQ. DIGITS(J)) GO TO 1
    2         CONTINUE
              IF(I .EQ. 1 ) RETURN
C***  NOW THE TOKEN IS AN IMPROPER IDENTIFIER
                      ALLDIG=3
                      RETURN
C*** TOKEN CONSISTS OF DIGITS ONLY
    3 ALLDIG=0
      RETURN
      END
C
C
      FUNCTION NGETC(C)
C
C     GET A (POSSIBLY PUSHED BACK) CHARACTER
C
      INTEGER BUF(100),BP,GETC,C,EOF
      COMMON /L3/ BUF,BP
      DATA BP/0/,EOF/2H*$/
C
      IF(BP .GT. 0) GO TO 1
              BP=1
              BUF(BP)=GETC(C)
              GO TO 2
```

```
    1   C=BUF(BP)
    2   IF(C .NE. EOF) BP=BP-1
              NGETC=C
              RETURN
        END
C
C
        SUBROUTINE PUTBAK(C)
C
C       PUSH CHARACTERS BACK ONTO INPUT
C
        INTEGER BUF(100),BP,BUFSIZ,C
        COMMON /L3/ BUF,BP
        DATA BUFSIZ/100/
C
        BP=BP+1
        IF (BP .GT. BUFSIZ) GO TO 1
              BUF(BP)=C
              RETURN
    1   ISN=10
        CALL ERRMES(ISN)
        STOP
        END
C
C
        INTEGER FUNCTION GETC(C)
C
C       THIS FUNCTION GETS CHARACTERS FROM STANDARD INPUT
C
        INTEGER GBUF(81),STAR,DOLLAR,C,EOF
        DATA LASTC/81/,STAR/1H*/,DOLLAR/1H$/,GBUF(81)/1H$/,EOF/2H*$/
C***    MAXLINE = MAXCARD + 1 = 80 + 1 = 81
  100   FORMAT(80A1)
        LASTC = LASTC + 1
        IF (LASTC .LE. 81) GO TO 1
              READ(5,100) (GBUF(I),I=1,80)
              CALL OUTRAT(GBUF)
              DO 3 I = 1,79
              IF(GBUF(I) .NE. STAR) GO TO 3
                    IF(GBUF(I+1) .EQ. DOLLAR) GO TO 2
    3         CONTINUE
              LASTC=1
    1   C=GBUF(LASTC)
        GETC=C
        RETURN
    2                C=EOF
                     GETC=EOF
                     RETURN
        END
C
C
```

```
      SUBROUTINE OUTTAB
C*** GET PAST COLUMN NO. 6
      INTEGER OUTBUF(81),OUTP,BLANK
      COMMON /L8/ OUTP,OUTBUF
      DATA BLANK/1H /
C
    1 IF(OUTP .GE. 6) RETURN
          CALL OUTCH(BLANK)
          GO TO 1
      END
C
C
      SUBROUTINE OUTSTR(LBUF,LEN)
C     OUTPUT STRING
      INTEGER LBUF(50),C,QUOTE
      DATA QUOTE/1H'/,LETH/1HH/
C
      I=1
    1 IF(I .GT. LEN) RETURN
          C=LBUF(I)
          IF(C .NE. QUOTE) GO TO 5
              I=I+1
              J=I
    2         IF(LBUF(J) .EQ. C) GO TO 3
                  J=J+1
                  GO TO 2
    3         JAYA=J-I
              CALL OUTNUM(JAYA)
              CALL OUTCH(LETH)
    4         IF(I .GE. J) GO TO 6
                  CALL OUTCH(LBUF(I))
                  I=I+1
                  GO TO 4
    5     CALL OUTCH(C)
    6     I=I+1
          GO TO 1
      END
C
C
      SUBROUTINE OUTNUM(N)
C     OUTPUT DECIMAL NUMBER,N
      INTEGER STR(11),C
      DATA STR/11*1H /
C
      LEN=ITOC(N,STR)
      DO 10 I = 1,LEN
        C=STR(I)
   10   CALL OUTCH(C)
      RETURN
      END
C
```

```
C
      SUBROUTINE OUTCON(N)
C     OUTPUT ' N  CONTINUE '
      INTEGER CONTIN(8)
      DATA CONTIN/1HC,1HO,1HN,1HT,1HI,1HN,1HU,1HE/
C
      IF(N .GT. 0) CALL OUTNUM(N)
      CALL OUTTAB
      CALL OUTSTR(CONTIN,8)
      CALL OUTDON
      RETURN
      END
C
C
      SUBROUTINE OUTGO(N)
C     OUTPUT ' GO TO N '
      INTEGER GOTO(4)
      DATA GOTO/1HG,1HO,1HT,1HO/
C
      CALL OUTTAB
      CALL OUTSTR(GOTO,4)
      CALL OUTNUM(N)
      CALL OUTDON
      RETURN
      END
C
C
      SUBROUTINE OUTCH(C)
C     PUT ONE CHARACTER INTO OUTPUT BUFFER
      INTEGER OUTBUF(81),OUTP,C,BLANK,STAR
      COMMON /L8/ OUTP,OUTBUF
      DATA BLANK/1H /,STAR/1H*/
C
      IF(OUTP .LT. 72) GO TO 3
          CALL OUTDON
          I=1
    1     IF(I .GE. 6) GO TO 2
              OUTBUF(I+9)=BLANK
              I=I+1
              GO TO 1
    2     OUTBUF(15)=STAR
          OUTP=6
    3 OUTP=OUTP+1
      OUTBUF(OUTP+9)=C
      RETURN
      END
C
C
      SUBROUTINE OUTDON
C     PRINT OUT AN OUTPUT LINE
      INTEGER OUTBUF(81),STR(11),OCTAL,OUTP,BLANK
```

```
      COMMON /L8/ OUTP,OUTBUF
      DATA LINFOR/0/,BLANK/1H /                                         RAD10031
C
  100 FORMAT(81A1)
      LINFOR=LINFOR+1
      LINE=OCTAL(LINFOR)
      LENGTH=ITOC(LINE,STR)
      DO 10 I = 1,LENGTH
   10 OUTBUF(I)=STR(I)
      NEXT=LENGTH+1
      DO 20 I = NEXT,9
   20 OUTBUF(I)=BLANK
      LAST=OUTP+10
      DO 30 I = LAST,81
   30 OUTBUF(I)=BLANK
      WRITE(1,100)  (OUTBUF(I),I=1,81)
      OUTP=0
      RETURN
      END
C
C
      SUBROUTINE OUTRAT(GBUF)
C*** OUTPUTS A LINE OF RATFOR SOURCE PROGRAM
      INTEGER RBUF(89),GBUF(81),STR(11),OCTAL,BLANK,SEMCOL,PREV
      COMMON /L5/ LINECT,SEMCOL,PREV
      DATA RBUF/89*1H /,BLANK/1H /
C
  101 FORMAT(25X,89A1)
      LINE=OCTAL(LINECT)
      LENGTH=ITOC(LINE,STR)
      DO 10 I =1,LENGTH
   10 RBUF(I)=STR(I)
      NEXT=LENGTH+1
      DO 20 I = NEXT,9
   20 RBUF(I)=BLANK
      DO 30 I = 10,89
      J=I-9
   30 RBUF(I)=GBUF(J)
      WRITE(6,101)  (RBUF(I),I=1,89)
      RETURN
      E 4
C
C
      IN3EGER FUNCTION OCTAL(NUM)
C     *** CONVERTS A DECIMAL NUMBER INTO AN OCTAL NUMBER
      INTEGER DEC,OCT,QUO,REM,EXP
C
      DEC=NUM
      OCT=0
      EXP=0
    1 QUO=DEC/8
```

```
      REM=DEC-QUO*8
      OCT=REM*10**EXP+OCT
      DEC=QUO
      EXP=EXP+1
      IF(QUO.NE.0) GO TO 1
      OCTAL=OCT
      RETURN
      END
C
C
      INTEGER FUNCTION ITOC(INT,STR)
C     CONVERTS AN INTEGER NUMBER INTO A CHARACTER STRING AND
C     RETURNS THE LENGTH OF THE STRING
      INTEGER STR(11)
C
  100 FORMAT(I1)
  101 FORMAT(A1)
      INTVAL=INT
      I=1
    1 IREM=INTVAL-INTVAL/10*10
      IF(IREM .EQ. 0) IREM=0
      REWIND 0
      WRITE(0,100) IREM
      REWIND 0
      READ(0,101) NUM
      STR(I)=NUM
      INTVAL=INTVAL/10
      I=I+1
      IF(INTVAL .NE. 0) GO TO 1
      I=I-1
      ITOC=I
      J=1
      IF(J .EQ. I) RETURN
    2 K=STR(I)
      STR(I)=STR(J)
      STR(J)=K
      I=I-1
      J=J+1
      IF(J .LT. I) GO TO 2
      RETURN
      END
C
C
      SUBROUTINE CTOI(STR,ID,NUM)
C     CONVERTS A STRING TO AN INTEGER
      INTEGER STR(ID)
C
  100 FORMAT(A1)
  101 FORMAT(I1)
      NUM=0
      I=1
```

```
1     NUM1=NUM
      REWIND 0
      WRITE(0,100) STR(I)
      REWIND 0
      READ(0,101) NUMBER
      NUM=NUM1*10+NUMBER
      I=I+1
      IF(I .LE. ID) GO TO 1
      RETURN
      END
C
C
      SUBROUTINE ERRMES(ISN)
C***  PRINTS OUT THE RELEVANT ERROR MESSAGE
      INTEGER LEXSTR(30),TOKSIZ,SEMCOL,PREV,OCTAL
      COMMON /L1/ LEXSTR,TOKSIZ
      COMMON /L5/ LINECT,SEMCOL,PREV
C
      LINE=OCTAL(LINECT)
      GOTO(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23)RAD10680
     1 ,ISN                                                          RAD10681
1     WRITE(6,101) LINE
      RETURN
2     WRITE(6,102) LINE  ,(LEXSTR(I),I=1,TOKSIZ)
      RETURN
3     WRITE(6,103) LINE
      RETURN
4     WRITE(6,104) LINE
      RETURN                                                         RAD10751
5     WRITE(6,105) LINE
      RETURN
6     WRITE(6,106) LINE
      RETURN
7     WRITE(6,107) LINE
      RETURN
8     WRITE(6,108) LINE
      RETURN
9     WRITE(6,109) LINE
      RETURN
10    WRITE(6,110) LINE
      RETURN
11    WRITE(6,111) LINE  ,LEXSTR(1)
      RETURN
12    WRITE(6,112) LINE
      RETURN
13    WRITE(6,113) LINE
      RETURN
14    WRITE(6,114) LINE
      RETURN
15    WRITE(6,115) LINE
      RETURN
```

```
 16   WRITE(6,116) LINE
      RETURN
 17   WRITE(6,117) LINE
      RETURN
 18   WRITE(6,118) LINE
      RETURN
 19   WRITE(6,119) LINE
      RETURN
 20   WRITE(6,120) LINE
      RETURN
 21   WRITE(6,121) LINE
      RETURN
 22   WRITE(6,122) LINE
      RETURN
 23   WRITE(6,123) LINE
      RETURN
101   FORMAT(1X,*WARNING - LINE NO. *,I3,* - TOKEN TOO LONG*)
102   FORMAT(1X,*ERROR - LINE NO. *,I3,* - POSSIBLE IMPROPER IDENTIFIER
     1 *,30A1)
103   FORMAT(1X,*ERROR - LINE NO. *,I3,* - ILLEGAL CHARACTER*)
104   FORMAT(1X,*ERROR - LINE NO. *,I3,* - ILLEGAL ELSE*)
105   FORMAT(1X,*ERROR - LINE NO. *,I3,* - ILLEGAL UNTIL*)
106   FORMAT(1X,*ERROR - LINE NO. *,I3,* - STACK OVERFLOW IN PARSER*)
107   FORMAT(1X,*ERROR - LINE NO. *,I3,* - ILLEGAL CLOSE*)
108   FORMAT(1X,*ERROR - LINE NO. *,I3,* - UNEXPECTED END-OF-FILE MARKE
     1R*)
109   FORMAT(1X,*WARNING - LINE NO. *,I3,* - POSSIBLE LABEL CONFLICT*)
110   FORMAT(1X,*WARNING - LINE NO. *,I3,* - MISSING LEFT PARENTHESIS*)
111   FORMAT(1X,*WARNING - LINE NO. *,I3,* - UNEXPECTED *,A1,* INSTEAD
     1OF LEFT PARENTHESIS*)
112   FORMAT(1X,*ERROR - LINE NO. *,I3,* - MISSING CONDITION PART IN FO
     1R STATEMENT*)
113   FORMAT(1X,*WARNING - LINE NO. *,I3,* - MISSING RIGHT PARENTHESIS*
     1)
114   FORMAT(1X,*ERROR - LINE NO. *,I3,* - ILLEGAL NEXT*)
115   FORMAT(1X,*ERROR - LINE NO. *,I3,* - ILLEGAL BREAK*)
116   FORMAT(1X,*ERROR - LINE NO. *,I3,* - UNEXPECTED BEGIN*)
117   FORMAT(1X,*ERROR - LINE NO. *,I3,* - UNBALANCED PARENTHESES*)
118   FORMAT(1X,*ERROR - LINE NO. *,I3,* - TOO MANY CHARACTERS PUSHED B
     1ACK*)
     1ACK*)
119   FORMAT(1X,*ERROR - LINE NO. *,I3,* - IDENTIFIER TABLE OVERFLOW*)
120   FORMAT(1X,*ERROR - LINE NO. *,I3,* - MISSING QUOTE*)
121   FORMAT(1X,*ERROR - LINE NO. *,I3,* - DEGREEOF FOR LOOP NESTING EX
     1CEEDS THE SPECIFIED LIMIT*)
122   FORMAT(1X,*ERROR - LINE NO. *,I3,* - UNEXPECTED SEMICOLON*)
123   FORMAT(1X,*ERROR - LINE NO. *,I3,* - STATEMENT NUMBER TOO LARGE*)
      END
```

ISN RATFOR SOURCE PROGRAM

```
/* THIS IS A DUMMY TEST PROGRAM
/*  USING ALL THE SPECIAL RATFOR CONSTRUCTS
INTEGER RADHA(10),ARRAY(20),CONSTANT,LIMIT,MAX,I,J
CHARACTER STRING(15),SPECIALONE,QUOTE,EOF,BUFFER(80)
/*
/*
FOR(I=1.,I.GT.MAX.,I=I+1) BEGIN
    RADHA(I)=ARRAY(I)*CONSTANT
    IF(RADHA(I).GT.LIMIT)
        BREAK
    IF(ARRAY(I).EQ.ARRAY(I+1))
        NEXT
    ELSE IF(ARRAY(I).EQ.0)
        ARRAY(I)=CONSTANT
CLOSE
I=1.,J=1
WHILE(STRING(I).NE.EOF) BEGIN
    BUFFER(J)=STRING(I)
    I=I+1
    J=J+1
CLOSE
REPEAT BEGIN
    IF(STRING(I).EQ.SPECIALONE)
        STRING(I)=BLANK
    ELSE IF(STRING(I).EQ.QUOTE)
        BREAK
CLOSE UNTIL(STRING(I).EQ.EOF)
        *$
```

0. 24 - UNEXPECTED END-OF-FILE MARKER

ISN TRANSLATED FORTRAN PROGRAM

```
      INTEGERRADHA(10),ARRAY(20),CONST1,LIMIT,MAX,I,J
      INTEGERSTRING(15),SPECI2,QUOTE,EOF,BUFFER(80)
      CONTINUE
      I=1
30000 IF(.NOT.(I.GT.MAX))GOTO30002
      RADHA(I)=ARRAY(I)*CONST1
      IF(.NOT.(RADHA(I).GT.LIMIT))GOTO30003
      GOTO30002
30003 CONTINUE
      IF(.NOT.(ARRAY(I).EQ.ARRAY(I+1)))GOTO30005
      GOTO30001
      GOTO30006
30005 CONTINUE
      IF(.NOT.(ARRAY(I).EQ.0))GOTO30007
      ARRAY(I)=CONST1
30007 CONTINUE
30006 CONTINUE
30001 I=I+1
      GOTO30000
30002 CONTINUE
```

```
      I=1
      J=1
      CONTINUE
30009 IF(.NOT.(STRING(I).NE.EOF))GOTO30010
      BUFFER(J)=STRING(I)
      I=I+1
      J=J+1
      GOTO30009
30010 CONTINUE
      CONTINUE
30011 CONTINUE
      IF(.NOT.(STRING(I).EQ.SPECI2))GOTO30013
      STRING(I)=BLANK
      GOTO30014
30013 CONTINUE
      IF(.NOT.(STRING(I).EQ.QUOTE))GOTO30015
      GOTO30012
30015 CONTINUE
30014 CONTINUE
      IF(.NOT.(STRING(I).EQ.EOF))GOTO30011
30012 CONTINUE
```

```
YS

 214000      TOTAL TIME          340483    (TIMES ARE IN MILLISECONDS)
   6416      DATA STORAGE          1641      AVAILABLE CORE        314      SYMBOL TABL
 DATE  081078

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Date Slip

A 55819

This book is to be returned on the date last stamped.
